2012

Depth Camera Based Indoor Mobile Robot Localization and Navigation

Joydeep Biswas
University of Massachusetts Amherst

Manuela M. Veloso
Carnegie Mellon University

Depth Camera Based Indoor Mobile Robot Localization and Navigation

Joydeep Biswas
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
joydeepb@ri.cmu.edu

Manuela Veloso
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
mmv@cs.cmu.edu

***Abstract*— The sheer volume of data generated by depth cameras provides a challenge to process in real time, in particular when used for indoor mobile robot localization and navigation. We introduce the Fast Sampling Plane Filtering (FSPF) algorithm to reduce the volume of the 3D point cloud by sampling points from the depth image, and classifying local grouped sets of points as belonging to planes in 3D (the "plane filtered" points) or points that do not correspond to planes within a specified error margin (the "outlier" points). We then introduce a localization algorithm based on an observation model that down-projects the plane filtered points on to 2D, and assigns correspondences for each point to lines in the 2D map. The full sampled point cloud (consisting of both plane filtered as well as outlier points) is processed for obstacle avoidance for autonomous navigation. All our algorithms process only the depth information, and do not require additional RGB data. The FSPF, localization and obstacle avoidance algorithms run in real time at full camera frame rates with low CPU requirements at more than 1030 frames per second on average. We provide experimental results demonstrating the effectiveness of our approach for indoor mobile robot localization and navigation. We further compare the accuracy and robustness in localization using depth cameras with FSPF *vs.* alternative approaches which simulate laser rangefinder scans from the 3D data.**

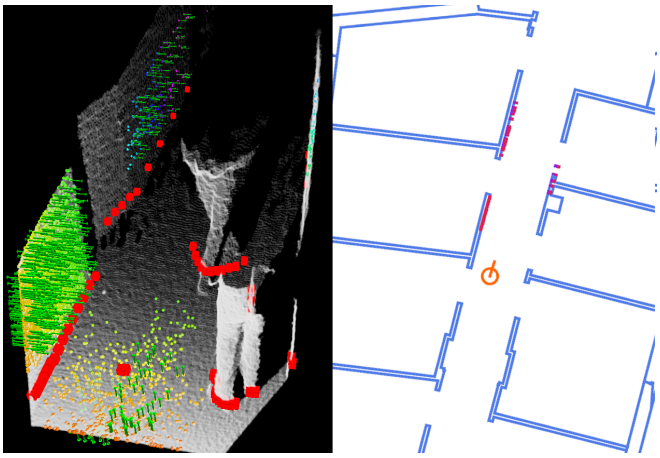


Fig. 1. Snapshot of depth image processing: On the left, the complete 3D point cloud is shown in white, the plane filtered 3D points in color along with plane normals, and the obstacle avoidance margins denoted by red boxes. On the right, the robot's pose is shown on the vector map (blue lines), with the 3D point correspondences shown as red points.

I. Introduction

The recent availability of inexpensive depth cameras has made available dense 3D point clouds, which were previously only accessible using much more expensive sensors like time-of-flight cameras or scanning 3D laser rangefinders. We are interested in using these depth cameras for ground based indoor mobile robots. We consider mobile robots with limited onboard computational power, and address two immediate challenges to using the depth cameras for mobile robot localization and navigation:

1) Depth cameras typically generate voluminous data that cannot be processed in its entirety in real time for localization (*e.g.*, the Microsoft Kinect sensor produces 9.2 million 3D pts/sec, compared to the 6800 2D pts/sec of the Hokuyo URG-04lx laser rangefinder).
2) Given that we already have existing 2D maps of our indoor environments, the observed 3D point clouds should be matched with the 2D maps.

In this paper, we tackle both these challenges. We first introduce the Fast Sampling Plane Filtering (FSPF) algorithm that samples the depth image to produce a set of points corresponding to planes, along with the plane parameters (normals and offsets). The volume of data to be processed is thus significantly reduced, addressing the first challenge with the additional advantage that non-planar objects in the scene, which are unlikely to correspond to map features are filtered. We then address the second challenge by introducing an observation model in our localization algorithm that matches the plane filtered points to the lines in the 2D maps, making it possible to reuse our existing 2D maps. This is in contrast to, and more effective than (as we shall show) down-projecting the 3D point cloud and binning it to simulate a conventional laser rangefinder. Our combined approach interestingly uses only the depth image and does not require the RGB images. The sampled points generated are also used to perform obstacle avoidance for navigation of the robot. Fig. 1 shows a snapshot of the key processed results after plane filtering, localization, and computing the obstacle avoidance margins.

Since our application is based on a ground robot, 6 degree of freedom (6D) localization is not required since the height of the sensor on the robot and its tilt and roll angles are fixed. At the same time, observations in 3D made by the depth camera has the potential to provide more useful information than planar laser rangefinders which sense objects only on a single 2D plane. Furthermore, typical indoor environments have an abundance of large planar features which are discernible in the depth images.

II. Related Work

Approaches that operate on raw 3D point clouds for plane (and general geometric shape) detection [1], [2] are ill-suited to running in real-time due to their high computational requirements, and because they ignore the fact that depth cameras make observations in "2.5D": the depth values are observed on a (virtual) 2D image plane originating from a single point. Region growing [3] exploits the local correlation in the depth image and attempts to assign planes to every 3D point. The Fast Sampling Plane Filtering algorithm, which we introduce in this paper, in contrast samples points at random and does not attempt to fit planes to every point, and instead uses local RANSAC [4].

There has been considerable work in 2D localization and mapping ([5] provides an overview of the recent advances in SLAM), and in using localization on 2D maps to generate 3D models using additional scans [6]. Specific to the problem of building 3D maps with 6 degrees of freedom (6D) localization is 6D SLAM [7], [8] that builds maps using 3D points in space, but these methods do not reason about the geometric primitives that the 3D points approximate.

An alternative approach to mapping using the raw 3D points is to map using planar features extracted from the 3D point cloud [9], [10]. In particular, 3D Plane SLAM [11] is a 6D SLAM algorithm that uses observed 3D point clouds to construct maps with 3D planes. The plane detection in their work relies on region growing [3] for plane extraction, whereas our approach uses sampling of the depth image. In addition, our observation model projects the observed planes onto the existing 2D vector map used for 2D laser rangefinder sensors.

More recently, techniques for 6D localization and mapping using RGB-D cameras have been explored. One such approach constructs surface element based dense 3D maps [12] which are simultaneously used for localizing in 3D using iterative closest point (ICP) as well as visual feature (SIFT) matching. While such approaches generate visually appealing dense 3D maps, they include in the maps features resulting from objects which are not likely to persist over time, like objects placed on tables, and the locations of movable chairs and tables.

In summary, the main contributions of this paper in relation to other work are:

- The Fast Sampling Plane Filtering algorithm that samples the depth image to produce a set of points corresponding to planes (Section III)
- A localization algorithm that uses this filtered point cloud to localize on a 2D vector map (Section IV)
- An obstacle avoidance algorithm that enables safe autonomous navigation (Section V)
- Experimental results (Section VI) showing the accuracy and reliability of FSPF based localization compared to the approach of localizing using simulated laser rangefinder scans, and long run autonomous trials of the robot using the depth camera alone.

III. Fast Sampling Plane Filtering

Depth cameras provide, for every pixel, color and depth values. This depth information, along with the camera intrinsics (horizontal field of view f_h, vertical field of view f_v, image width w and height h in pixels) can be used to reconstruct a 3D point cloud. Let the depth image of size $w \times h$ pixels provided by the camera be I, where $I(i,j)$ is the depth of a pixel at location $d = (i,j)$. The corresponding 3D point $p = (p_x, p_y, p_z)$ is reconstructed using the depth value $I(d)$ as

$$p_x = I(d)\left(\frac{j}{w-1} - 0.5\right)\tan\left(\frac{f_h}{2}\right), \quad (1)$$

$$p_y = I(d)\left(\frac{i}{h-1} - 0.5\right)\tan\left(\frac{f_v}{2}\right), \quad (2)$$

$$p_z = I(d). \quad (3)$$

With limited computational resources, most algorithms (*e.g.* localization, mapping *etc.*) cannot process the full 3D point cloud at full camera frame rates in real time. The naïve solution would therefore be to sub-sample the 3D point cloud for example, by dropping (say) one out of three points, or sampling randomly. Although this reduces the number of 3D points being processed by the algorithms, it ends up discarding information about the scene. An alternative solution is to convert the 3D point cloud into a more compact, feature - based representation, like planes in 3D. However, computing optimal planes to fit the point cloud for every observed 3D point would be extremely CPU-intensive and sensitive to occlusions by obstacles which exist in real scenes. The Fast Sampling Plane Filtering (FSPF) algorithm combines both ideas: it samples random neighborhoods in the depth image, and in each neighborhood, it performs a RANSAC based plane fitting on the 3D points. Thus, it reduces the volume of the 3D point cloud, it extracts geometric features in the form of planes in 3D, and it is robust to outliers since it uses RANSAC within the neighborhood.

FSPF takes the depth image I as its input, and creates a list P of n 3D points, a list R of corresponding plane normals, and a list O of outlier points that do not correspond to any planes. Algorithm 1 outlines the plane filtering procedure. It uses the helper subroutine $[\text{numInliers}, \hat{P}, \hat{R}] \leftarrow \text{RANSAC}(d_0, w', h', l, \epsilon)$, which performs the classical RANSAC algorithm over the window of size $w' \times h'$ around location d_0 in the depth image, and returns inlier points and normals $\hat{P}$ and $\hat{R}$ respectively, as well as the number of inlier points found. The configuration parameters required by FSPF are listed in Table I.

FSPF proceeds by first sampling three locations d_0,d_1,d_2 from the depth image (lines 9-11). The first location d_0 is selected randomly from anywhere in the image, and d_1 and d_2 are selected randomly within a neighborhood of size η around d_0. The 3D coordinates for the corresponding points p_0, p_1, p_2 are then computed using eq. 1-3. A search window of width w' and height h' is computed based on the mean depth (z-coordinate) of the points p_0, p_1, p_2 (lines 14-16), and the minimum expected size S of the planes in the world.

Algorithm 1 Fast Sampling Plane Filtering

procedure PLANEFILTERING(I)
  $P \leftarrow \{\}$ ▷ Plane filtered points
  $R \leftarrow \{\}$ ▷ Normals to planes
  $O \leftarrow \{\}$ ▷ Outlier points
  $n \leftarrow 0$ ▷ Number of plane filtered points
  $k \leftarrow 0$ ▷ Number of neighborhoods sampled
  while $n < n_{max} \wedge k < k_{max}$ **do**
    $k \leftarrow k + 1$
    $d_0 \leftarrow (\text{rand}(0, h-1), \text{rand}(0, w-1))$
    $d_1 \leftarrow d_0 + (\text{rand}(-\eta, \eta), \text{rand}(-\eta, \eta))$
    $d_2 \leftarrow d_0 + (\text{rand}(-\eta, \eta), \text{rand}(-\eta, \eta))$
    Reconstruct p_0, p_1, p_2 from d_0,d_1,d_2
    $r = \frac{(p_1 - p_0) \times (p_2 - p_0)}{||(p_1 - p_0) \times (p_2 - p_0)||}$ ▷ Compute plane normal
    $\bar{z} = \frac{p_{0z} + p_{1z} + p_{2z}}{3}$
    $w' = w\frac{S}{\bar{z}}\tan(f_h)$
    $h' = h\frac{S}{\bar{z}}\tan(f_v)$
    $[\text{numInliers}, \hat{P}, \hat{R}] \leftarrow \text{RANSAC}(d_0, w', h', l, \epsilon)$
    if numInliers $> \alpha_{in} l$ **then**
      Add $\hat{P}$ to P
      Add $\hat{R}$ to R
      numPoints $\leftarrow$ numPoints + numInliers
    else
      Add $\hat{P}$ to O
    end if
  end while
  return P, R, O
end procedure

Local RANSAC is then performed in the search window. If more than $\alpha_{in} l$ inlier points are produced as a result of running RANSAC in the search window, then all the inlier points are added to the list P, and the associated normals to the list R. This algorithm is run a maximum of m_{max} times to generate a list of maximum n_{max} 3D points and their corresponding plane normals. Fig. 2 shows an example scene with the plane filtered points and their corresponding plane normals.

Parameter	Value	Description
n_{max}	2000	Maximum total number of filtered points
k_{max}	20000	Maximum number of neighborhoods to sample
l	80	Number of local samples
η	60	Neighborhood for global samples (in pixels)
S	$0.5m$	Plane size in world space for local samples
ϵ	$0.02m$	Maximum plane offset error for inliers
α_{in}	0.8	Minimum inlier fraction to accept local sample

TABLE I
CONFIGURATION PARAMETERS FOR FSPF

IV. LOCALIZATION

For the task of localization, the plane filtered point cloud P and the corresponding plane normal estimates R need to be related to the 2D map. The 2D map representation which we use is a "vector" map: it represents the environment as a set of line segments (corresponding to the obstacles in the environment), as opposed to the more commonly used occupancy grid [13] based maps. The observation model therefore has to compute the line segments likely to be observed by the robot given its current pose and the map. This is done by an analytic ray cast step. We therefore introduce next the representation of the 2D vector map and the algorithm for analytic ray casting using the 2D vector map.

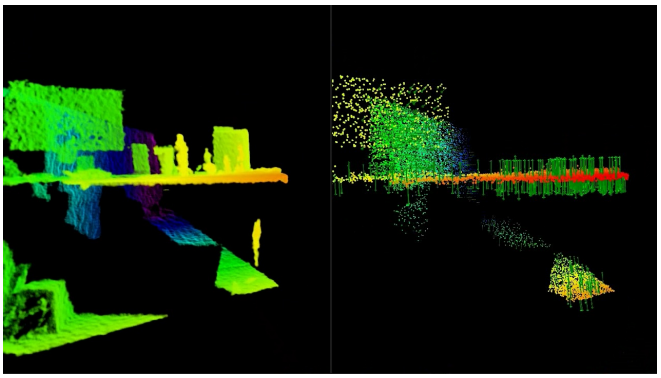


Fig. 2. Fast Sampling Plane Filtering in a scene with a cluttered desktop. The complete 3D point cloud is shown on the left, the plane filtered points and the corresponding normals on the right. The table clutter is rejected by FSPF while preserving the large planar elements like the monitors, the table surface, the walls and the floor.

A. Vector Map Representation and Analytic Ray Casting

The map M used by our localization algorithm is a set of s line segments l_i corresponding to all the walls in the environment: $M = \{l_i\}_{i=1:s}$. Such a representation may be acquired by mapping (*e.g.* [14]) or (as in our case) taken from the blueprints of the building.

Given this map, to compute the observation likelihoods based on observed planes, the first step is to estimate which lines on the map are likely to be observed (the "scene lines"), given the pose estimate of the robot. This ray casting step is analytically computed using the vector map representation.

The procedure to analytically generate a ray cast at location x given the map M is outlined in Algorithm 2. The returned result is the scene lines L: a list of non-intersecting, non-occluded line segments visible by the robot from the location x. This algorithm calls the helper procedure $\text{TrimOcclusion}(x, l_1, l_2, L)$ that accepts a location x, two lines l_1 and l_2 and a list of lines L. TrimOcclusion trims line l_1 based on the occlusions due to the line l_2 as seen from the location x. The list L contains lines that yet need to be tested for occlusions by l_2. There are in general 4 types of arrangements of l_1 and l_2, as shown in Fig. 3:

1) l_1 is not occluded by l_2. In this case, l_1 is unchanged.
2) l_1 is completely occluded by l_2. l_1 is trimmed to zero length by TrimOcclusion.
3) l_1 is partially occluded by l_2. l_1 is first trimmed to a non occluded length, and if a second disconnected non occluded section of l_1 exists, it is added to L.
4) l_1 intersects with l_2. Again, l_1 is first trimmed to an non occluded length, and if a second disconnected non occluded section of l_1 exists, it is added to L.

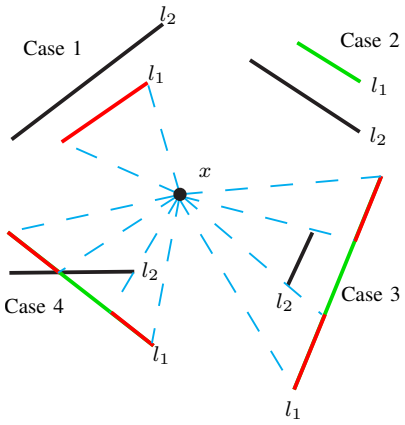


Fig. 3. Line occlusion cases. Line l_1 is being tested for occlusion by line l_2 from location x. The occluded parts of l_1 are shown in green, and the visible parts in red. The visible ranges are bounded by the angles demarcated by the blue dashed lines.

Algorithm 2 Analytic Ray Cast Algorithm

1: **procedure** ANALYTICRAYCAST(M, x)
2: $\hat{L} \leftarrow M$
3: $L \leftarrow \{\}$
4: **for** $l_i \in \hat{L}$ **do**
5: **for** $l_j \in L$ **do**
6: TrimOcclusion$(x, l_i, l_j, \hat{L})$
7: **end for**
8: **if** $||l_i|| > 0$ **then** ▷ l_i is partly non occluded
9: **for** $l_j \in L$ **do**
10: TrimOcclusion$(x, l_j, l_j, \hat{L})$
11: **end for**
12: $L \leftarrow L \cup \{l_i\}$
13: **end if**
14: **end for**
15: **return** L
16: **end procedure**

The analytic ray casting algorithm (Algorithm 2) proceeds as follows: A list $\hat{L}$ of all possible lines is made from the map M. Every line $l_i \in \hat{L}$ is first trimmed based on occlusions by lines in the existing scene list L (lines 5-7). If at least part of l_1 is left non occluded, then the existing lines in $\hat{L}$ are trimmed based on occlusions by l_i (lines 9-11) and l_i is then added to the scene list L. The result is a list of non occluded, non-intersecting scene lines in L. Fig. 4 shows an example scene list on the real map.

Thus, given the robot pose, the set of line segments likely to be observed by the robot is computed. Based on this list of line segments, the actual observation of the plane filtered point cloud P is related to the map using the projected 3D point cloud model, which we introduce next.

B. Projected 3D Point Cloud Observation Model

Since the map on which the robot is localizing is in 2D, the 3D filtered point cloud P and the corresponding plane normals R are first projected onto 2D to generate a 2D point cloud P' along with the corresponding normalized normals

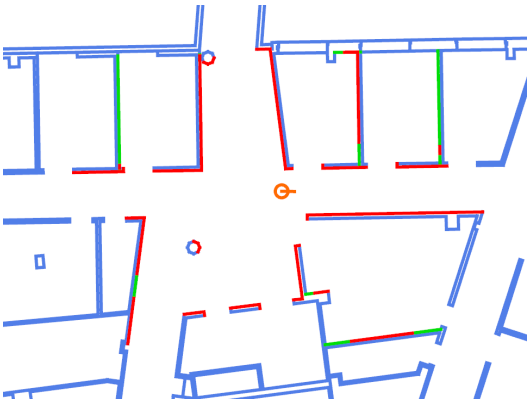


Fig. 4. Analytically rendered scene list. The lines in the final scene list L are shown in red, the original, untrimmed corresponding lines in green, and all other lines on the map in blue.

R'. Points that correspond to ground plane detections are rejected at this step. Let the pose of the robot x be given by $x = \{x_1, x_2\}$ where x_1 is the 2D location of the robot, and x_2 its orientation angle. The observable scene lines list L is computed using an analytic ray cast. The observation likelihood $p(y|x)$ (where the observation y is the 2D projected point cloud P') is computed as follows:

1) For every point p_i in P', line l_i ($l_i \in L$) is found such that the ray in the direction of $p_i - x_1$ and originating from x_1 intersects l_i.
2) Points for which no such line l_i can be found are discarded.
3) Points p_i for which the corresponding normal estimates r_i differ from the normal to the line l_i by a value greater than a threshold θ_{max} are discarded.
4) The perpendicular distance d_i of p_i from the (extended) line l_i is computed.
5) The total (non-normalized) observation likelihood $p(y|x)$ is then given by:

$$p(y|x) = \prod_{i=1}^{n} \exp\left[-\frac{d_i^2}{2f\sigma^2}\right] \tag{4}$$

Here, σ is the standard deviation of a single distance measurement, and $f : f > 1$ is a discounting factor to discount for the correlation between rays. The observation likelihoods thus computed are used for localization using the Corrective Gradient Refinement (CGR) [15] algorithm, which we review in brief.

C. Corrective Gradient Refinement for Localization

The belief of the robot's location is represented as a set of weighted samples or "particles", as in Monte Carlo Localization (MCL)[16]: $Bel(x_t) = \{x_t^i, w_t^i\}_{i=1:m}$. The CGR algorithm iteratively updates the past belief $Bel(x_{t-1})$ using observation y_t and control input u_{t-1} as follows:

1) Samples of the belief $Bel(x_{t-1})$ are evolved through the motion model, $p(x_t|x_{t-1}, u_{t-1})$ to generate a first stage proposal distribution q^0.
2) Samples of q^0 are "refined" in r iterations (which produce intermediate distributions $q^i, i \in [1, r-i]$)

using the gradients $\frac{\delta}{\delta x}p(y_t|x)$ of the observation model $p(y_t|x)$.

3) Samples of the last generation proposal distribution q^r and the first stage proposal distribution q^0 are sampled using an acceptance test to generate the final proposal distribution q.
4) Samples x_t^i of the final proposal distribution q are weighted by corresponding importance weights w_t^i, and resampled with replacement to generate $Bel(x_t)$.

Therefore, for CGR we need to compute both the observation likelihood, as well as its gradients. The observation likelihood is computed using Eq. 4, and the corresponding gradients are therefore given by,

$$\frac{\delta}{\delta x}p(y|x) = -\frac{p(y|x)}{f\sigma^2}\sum_{i=1}^{n}\left[d_i\frac{\delta d_i}{\delta x}\right]. \tag{5}$$

The term $\frac{\delta d_i}{\delta x}$ in this equation has two terms, corresponding to the translation component $\frac{\delta d_i}{\delta x_1}$ and the rotational component $\frac{\delta d_i}{\delta x_2}$. These terms are computed by rigid body translation and rotation of the point cloud P respectively.

The observation likelihoods and their gradients thus computed are used to update the localization using CGR.

V. Navigation

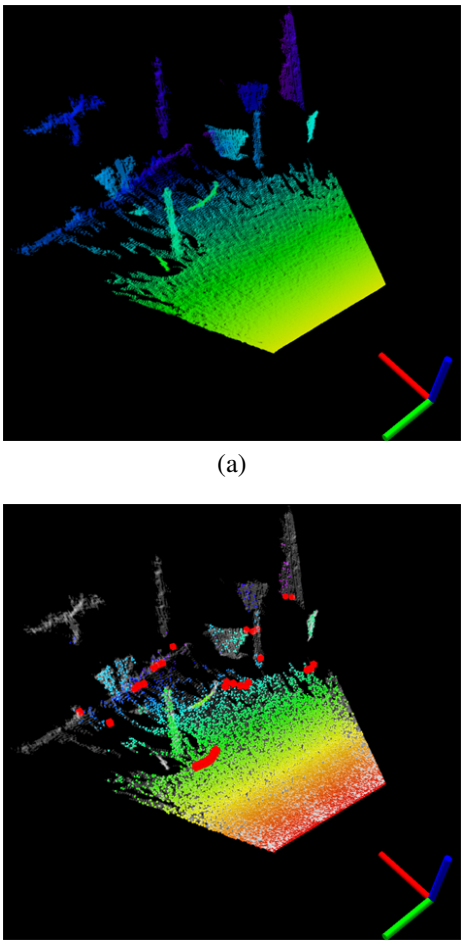
For the robot to navigate autonomously, it needs to be able to successfully avoid obstacles in its environment. This is done by computing open path lengths available to the robot for different angular directions. Obstacle checks are performed using the 3D points from the sets P and O. Given the robot radius r and the desired direction of travel θ_d, the open path length $d(\theta)$ as a function of the direction of travel θ, and hence the chosen obstacle avoidance direction θ^* are calculated as:

$$P_\theta = \left\{p : p \in P \cup O \wedge ||p - p\cdot\hat{\theta}|| < r\right\} \tag{6}$$

$$d(\theta) = \min_{p\in P_\theta}\left(\max(0, ||p\cdot\hat{\theta}|| - r)\right) \tag{7}$$

$$\theta^* = \arg\max_\theta\left(d(\theta)\cos(\theta - \theta_d)\right) \tag{8}$$

Here, $\hat{\theta}$ is a unit vector in the direction of the angle θ, and the origin of the coordinate system is coincident with the robot's center. Fig. 5 shows an example scene with two tables and four chairs that are detected by the depth camera. Despite randomly sampling (with a maximum of 2000 points) from the depth image, all the obstacles are correctly detected, including the table edges. The computed open path lengths from the robot location are shown by red boxes.



(a)

(b)

Fig. 5. Obstacle avoidance: The raw 3D point cloud (a) and (b) the sampled points (shown in color), along with the open path limits (red boxes). The robot location is marked by the axes.

VI. Experimental Results

We evaluated the performance of our depth camera based localization and navigation algorithms over two sets of experiments. The first set of experiments compare our approach using FSPF CGR localization to three other approaches that use the Kinect for localization by simulating laser rangefinder scans from the Kinect sensor. The second set of long run trials test the effectiveness of our complete localization and navigation system over a long period of time.

Our experiments were performed on our custom built omnidirectional indoor mobile robot, equipped with the Microsoft Kinect sensor. The Kinect sensor provides depth images of size 640×480 pixels at 30Hz. To compare the accuracy in localization of the different approaches using the Kinect, we also used a Hokuyo URG-04LX 2D laser rangefinder scanner as a reference. The autonomous long run trials were run using the Kinect alone for localization and navigation. All trials were run single threaded, on a single core of an Intel Core i7 950 processor.

A. Comparison of FSPF to Simulated Laser Rangefinder localization

We compared our approach using FSPF CGR localization to the following three other CGR based laser rangefinder localization algorithms where the data from the Kinect sensor was used to simulate laser rangefinder scans:

1) Extracting a single raster line from the Kinect depth image, reconstructing the corresponding 3D points, and

then down-projecting into 2D to generate the simulated laser rangefinder scans. We call this approach the Kinect-Raster (KR) approach.

2) Randomly sampling locations in the Kinect depth image, and using the corresponding 3D points to simulate the laser rangefinder scan. We call this approach the Kinect-Sampling (KS) approach.
3) Reconstructing the full 3D point cloud from the entire Kinect depth image, and using all these points to generate the simulated laser rangefinder scan. We call this approach the Kinect-Complete (KC) approach.

For estimating the error in localization using the Kinect, we used the localization estimates produced by the laser rangefinder CGR localization algorithm for reference. In the KR, KS and KC approaches, the simulated laser rangefinder scan had a simulated angular range of 180° and an angular resolution of 0.35°, although only part of this scan was populated with useful information due to the limited angular field of view of the Kinect. The number of points sampled in the KS approach was limited to 2000 points, the same as the number of plane filtered points generated in the FSPF approach.

We recorded a log with odometry and data from the Kinect sensor and the laser rangefinder while traversing a path consisting of corridors as well as open areas with unmapped obstacles. There was significant human traffic in the environment, with some humans deliberately blocking the path of the robot. This log was replayed offline for the different localization approaches, running 100 times per approach, with randomly added 20% noise to the odometry data. Fig. 6 shows the combined traces of the localization estimates for all the succesful trials of each of the approaches. A trial was said to be "succesful" if the error in localization was less than $1m$ at every timestep of the trial.

Fig. 8 shows a cumulative histogram of the error in localization using the different approaches. The FSPF approach has significantly less error than the other three approaches. Fig. 7 shows the cumulative failure rate as a function of the elapsed run time. The FSPF approach has a 2% failure rate at the end of all the trials, wehereas all the other approaches start failing after around $20s$ into the trials. The KR, KS, and KC approaches have total failure rates of 82%, 62% and 61% respectively. The abrupt increase in failures around the $20s$ mark corresponds to the time when the robot encounters unmapped objects in the form of humans, tables and chairs in an open area.

To compare the execution times of the different approaches, we kept track of the time taken to process all the Kinect depth image observations, and calculated the ratio of these execution times to the total duration of the trials. These values are thus indicative of the mean CPU processing load while running the algorithms online on the robot in real time. The values for the different algorithms were 0.01%, 3.6%, 56.6% and 16.3% respectively for the KR, KS, KC, and FSPF approaches respectively. It should be noted that since the KR, KS and KC approaches use simulated laser rangefinder scans with a fixed (180) number of rays, while the FSPF algorithm uses as many plane filtered points as are detected up to a maximum of n_{max}, which we set to 2000 for the experiments.

Fig. 6. Combined traces of all successful trials of all approaches: green (KR), red (KS), blue (KC) and black (FSPF). FSPF CGR localization is seen to have the least variation across trials.

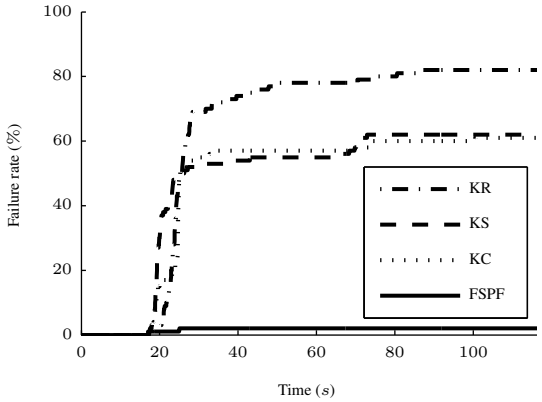


Fig. 7. Cumulative fractions of the failure rates as a function of elapsed time for all four approaches

B. Long Run Trials

To test the robustness of the depth-camera based FSPF localization and navigation solution, we set a series of random waypoints for the robot to navigate to, spread across the map. The total length of the path was just over 4km. Over the duration of the experiment, only the Kinect sensor was used for localization and obstacle avoidance. The robot successfully navigated to all waypoints, but localization had to be reset at three locations, which were in open areas of the map with unmapped obstacles where Kinect sensor could not observe any walls for a while. Fig. 9 shows the trace of

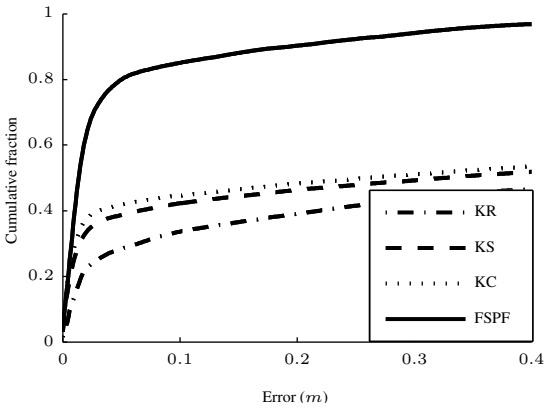


Fig. 8. Cumulative histogram of errors in localization of the different approaches

the robot's location over the course of the experiment. We continue to use the FSPF localization and obstacle avoidance algorithms during daily deployments of our robot.

VII. Conclusion And Future Work

In this paper, we introduced an algorithm for efficient depth camera based localization and navigation for indoor mobile robots. We introduced the Fast Sampling Plane Filtering algorithm to filter depth images into point clouds corresponding to local planes. We subsequently contributed an observation model that matches the plane filtered points to lines in the existing 2D maps for localization. Both the plane filtered, as well as the outlier point clouds are further used for obstacle avoidance. We experimentally showed FSPF localization to be more accurate as well as more robust compared to localization using Kinect based simulated laser rangefinder readings. We further demonstrated a long run trial of the robot autonomously operating for over 4km using the depth camera alone.

For use on other platforms like UAVs, scaling up the state space to full 6 degrees of freedom is another possible avenue of future work.

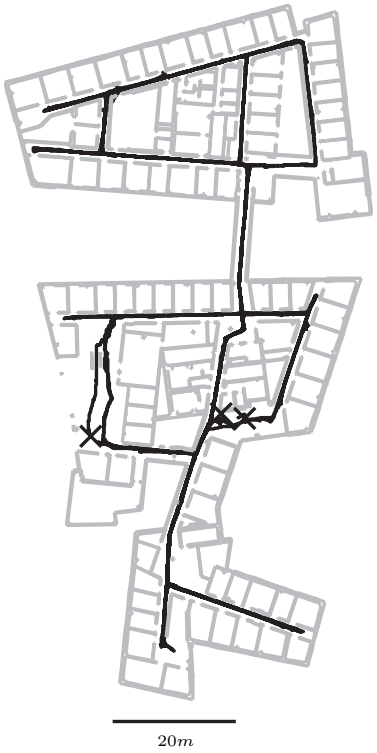


Fig. 9. Trace of robot location for the long run trial. The locations where localization had to be reset are marked with crosses.

References

[1] N.J. Mitra and A. Nguyen. Estimating surface normals in noisy point cloud data. In *Proceedings of the nineteenth annual symposium on Computational geometry*, pages 322–328. ACM, 2003.

[2] R. Schnabel, R. Wahl, and R. Klein. Efficient RANSAC for Point-Cloud Shape Detection. In *Computer Graphics Forum*, volume 26, pages 214–226. Wiley Online Library, 2007.

[3] J. Poppinga, N. Vaskevicius, A. Birk, and K. Pathak. Fast plane detection and polygonalization in noisy 3D range images. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3378–3383. IEEE, 2008.

[4] M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395.

[5] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *Robotics & Automation Magazine, IEEE*, 13(2):99–110, 2006.

[6] D. Hähnel, W. Burgard, and S. Thrun. Learning compact 3D models of indoor and outdoor environments with a mobile robot. *Robotics and Autonomous Systems*, 44(1):15–27, 2003.

[7] A. Nuchter, K. Lingemann, J. Hertzberg, and H. Surmann. 6d SLAM with approximate data association. In *Advanced Robotics, 2005. ICAR'05. Proceedings., 12th International Conference on*, pages 242–249. IEEE, 2005.

[8] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann. 6D SLAM - 3D mapping outdoor environments. *Journal of Field Robotics*, 24(8-9):699–722, 2007.

[9] J. Weingarten and R. Siegwart. 3D SLAM using planar segments. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3062–3067. IEEE, 2006.

[10] P. Kohlhepp, P. Pozzo, M. Walther, and R. Dillmann. Sequential 3D-SLAM for mobile action planning. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 1, pages 722–729. IEEE, 2004.

[11] K. Pathak, A. Birk, N. Vaskevicius, M. Pfingsthorn, S. Schwertfeger, and J. Poppinga. Online three-dimensional SLAM by registration of large planar surface segments and closed-form pose-graph relaxation. *Journal of Field Robotics*, 27(1):52–84, 2010.

[12] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using depth cameras for dense 3d modeling of indoor environments. In *the 12th International Symposium on Experimental Robotics*, 2010.

[13] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[14] L. Zhang and B.K. Ghosh. Line segment based map building and localization using 2D laser rangefinder. In *IEEE Int. Conf. on Robotics and Automation*, 2000.

[15] J. Biswas, B. Coltin, and M. Veloso. Corrective gradient refinement for mobile robot localization. In *Intelligent Robots and Systems (IROS), 2011 IEEE International Conference on*. IEEE, 2011.

[16] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the National Conference on Artificial Intelligence*, pages 343–349. JOHN WILEY & SONS LTD, 1999.